CENTRE DE RECHERCHE
SUR LES TRANSPORTS
(C.R.T.)
CENTRE FOR RESEARCH
ON TRANSPORTATION

UNIVERSITÉ DE MONTRÉAL
ÉCOLE DES HAUTES ÉTUDES COMMERCIALES
ÉCOLE POLYTECHNIQUE

CENTRE DE RECHERCHE SUR LES TRANSPORTS
UNIVERSITÉ DE MONTRÉAL
C.P. 6128, SUCCURSALE A
MONTRÉAL
CANADA H3C 3J7

TÉLÉPHONE : (514) 343-7575
EMAIL : crt@crt.umontreal.ca
TÉLÉCOPIEUR (FAX) : (514) 343-7121

92-07826/

CRT-841

A Composite Heuristic for the Identical Parallel Machine Scheduling Problem with Minimum Makespan Objective

by

Paulo M. França[1]
Michel Gendreau[2]
Gilbert Laporte[2]
Felipe M. Müller[1,2,3]

[1] Faculdade de Engenharia Elétrica, Universidade Estadual de Campinas, Brazil.
[2] Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale A, Montréal, Canada H3C 3J7
[3] Núcleo Processamento de Dados, Universidade Federal de Santa Maria, Brazil.

Centre de recherche sur les transports — Publication #841

July 1992

Dépôt légal – Bibliothèque nationale du Québec,
Bibliothèque nationale du Canada, 1992

ABSTRACT

This paper describes a new heuristic algorithm for the problem of scheduling n independent jobs on m identical parallel machines in order to minimize the makespan. The algorithm produces near-optimal solutions in short computation times. It outperforms alternative heuristics on a series of test problems.

Key words : machine scheduling.

- - - - - - - - - - - -

RÉSUMÉ

On décrit dans cet article un nouvel algorithme heuristique pour le problème d'ordonnancement de n tâches indépendantes sur m machines parallèles identiques, où l'objectif est de minimiser le temps de complétion global. L'algorithme produit des solutions quasi optimales moyennant des temps de calcul très courts. L'algorithme se compare avantageusement à d'autres méthodes sur une série de problèmes tests.

Mots-clés : ordonnancement.

1. Introduction

In this article, we propose a new heuristic procedure for the problem of scheduling n independent *jobs* $J_1, ..., J_n$ on m identical parallel *machines* $M_1, ..., M_m$ without preemption, where $n \geq m \geq 2$ and each job J_j has a positive processing time p_j. Let C_i be the completion time on machine M_i. Then the problem is to determine a schedule minimizing the *makespan* $C_{\max} = \max_{i=1,...,m} \{C_i\}$. Using the three-field classification scheme introduced by Graham, Lawler, Lenstra and Rinnooy Kan [1], this problem is denoted as $P||C_{\max}$.

The object of this paper is to present 3–PHASE, a composite heuristic consisting of a construction phase, followed by two improvement phases. This algorithm is described in Section 2. Computational results, presented in Section 3, indicate that the proposed algorithm produces near-optimal solutions and compares favourably with alternative heuristics.

The $P||C_{\max}$ problem is one of the most studied problems in the field of scheduling theory. Surveys have been provided by Lawler, Lenstra, Rinnooy Kan and Shmoys [2], and by Cheng and Sin [3]. Garey and Johnson [4] have proved the problem is NP-hard in the strong sense. Optimal algorithms have been proposed by Blazewicz [5] and by Dell'Amico and Martello [6]. In addition, Bratley, Florian and Robillard [7] describe an enumerative procedure for a generalization of the problem in which jobs have earliest start and due date constraints.

Most algorithms developed for $P||C_{\max}$ are heuristics. These can be broadly classified into *constructive heuristics* and *improvement heuristics*. The majority of algorithms belong to the first category and have a known worst-case performance

ratio (see, e.g., Graham [8] Coffman, Garey and Johnson [9], Friesen [10], Friesen and Langston [11], Sahni [12], Hochbaum and Shmoys [13], and Leung [14]). A well known constructive heuristic is the so-called LPT (*Longest Processing Time*) method due to Graham [8]. Here the jobs are sorted and relabelled so that $p_1 \geq p_2 \geq \ldots \geq p_n$. Starting with job J_1, all jobs are then successively assigned to the least loaded machine. The worst-case ratio of LPT is equal to $4/3 - 1/(3m)$, and the time complexity of the algorithm is equal to $O(n \log n)$. Numerical tests have shown that LPT has a good empirical performance, especially for large values of n (see Finn and Horowitz [15]). Another classical constructive heuristic is the *Multifit* (MF) algorithm of Coffman, Garey and Johnson [9]. Here, one seeks to determine the smallest machine capacity so that a feasible solution can be found using the LPT scheme. This is achieved by solving heuristically a series of *Bin Packing Problems* (see, e.g., Coffman, Garey and Johnson [16]). It is shown that if k packing attempts are made, MF can run in $O(n \log n + kn \log n)$ time and the worst-case ratio of the algorithm does not exceed $11/9 + 2^{-k}$. Finn and Horowitz [15] have shown that the effectiveness of MF is comparable to that of LPT, even though MF has a worse time efficiency. In improvement heuristics, job reassignments and interchanges are attempted to improve a feasible solution obtained by any means. The $0/1$ interchange algorithm of Finn and Horowitz [15] is of this type; it has a worst-case performance ratio of 2. An improved procedure with a bound of 4/3 was later proposed by Langston [17].

2. The 3–PHASE Algorithm

The proposed 3–PHASE algorithm can be summarized as follows. In Phase 1, jobs are classified according to their processing times and assigned to machines in order to achieve a fairly balanced load. In Phase 2, the workload balance is improved by repeatedly moving a job from the most loaded to the least loaded machine. Finally in Phase 3, an even better balance is achieved by interchanging jobs between machines. We now describe the three phases.

Phase 1 : ***Initial Assignment.*** Contrary to several known constructive algorithm, it is not necessary in this constructive phase to sort the jobs in order of processing times. Only $\underline{p} = \min\{p_j\}$ and $\overline{p} = \max\{p_j\}$ are required. Then, $[\underline{p}, \overline{p}]$ is partitioned into r equal intervals $I_1, ..., I_r$, where r is an input parameter of the algorithm. Job J_j is called an ℓ–job if $p_j \in I_\ell$. First initialize $s_1 := ... = s_r := 0$, the index of the machine to which the last ℓ–job has been assigned ($\ell = 1, ..., r$). Then consecutively consider all jobs J_j: if J_j is an ℓ–job, assign it to machine $s_\ell \pmod m + 1$ and set $s_\ell := s_\ell \pmod m + 1$.

Phase 2 : ***Job Reassignments.*** Let $\overline{C} = \sum_{i=1}^{m} C_i/m$ be the average machine completion time; this is independent of the particular assignment of jobs to machines. This step consists of repeatedly moving a job from the busiest machine (with the largest C_i), to the least busy machine in order to reduce the makespan. To ensure convergence, the processing time of any machine always remains on the same side of $\overline{C}$. The procedure is now described.

Step 1. Identify the most and least busy machines and relabel them M_1 and M_m, respectively.

Step 2. Compute $\Delta_1 = C_1 - \overline{C}$, $\Delta_m = \overline{C} - C_m$, $\Delta = \min\{\Delta_1, \Delta_m\}$. If $\Delta < \underline{p}$, Phase 2 terminates. Otherwise, Δ belongs to some interval I_ℓ.

Step 3. If no job J_j with $p_j \in I_\ell$ and $p_j \leq \Delta$ is assigned to M_1, go to Step 4. Otherwise, reassign a job of I_ℓ from M_1 to M_m and go to Step 1.

Step 4. If no job J_j with $p_j \in I_\ell^- = I_1 \cup ... \cup I_{\ell-1}$ is assigned to M_1, Phase 2 terminates. Otherwise, reassign a job J_j with $p_j \in I_\ell^-$ from M_1 to M_m and go to Step 1.

Two remarks are in order. First, the use of $\overline{C}$ as a "target" to guide the job reassignment is advantageous when compared with the standard procedure used in [15], since machines with completion times in the neighbourhood of $\overline{C}$ are unlikely to be selected for reassignments. Second, the fact that jobs are grouped in subsets according to their processing times provides for a more efficient search of candidate jobs for reassignment. Tests have shown that in Steps 3 or 4, there is no real advantage, when doing reassignment, in looking for the job J_j with the best fit, i.e., with the largest p_j not exceeding Δ; any job with $p_j \leq \Delta$ will do.

Phase 3. Job Interchanges. Again, M_1 and M_m refer to the two machines with heaviest and lightest workload, respectively. (The remaining machines are labeled arbitrarily). In this phase we attempt to interchange a job from M_1 (job J_j, say) with a job from another machine M_h (job $J_{j'}$, say), starting with $h = m, m-1, ...,$ until an advantageous exchange has been identified, i.e., an exchange such that $\theta = p_j - p_{j'} < C_1 - C_h = d$. The idea is to use the value of $d/2$ as a target for the difference in processing times between two interchanged jobs, very much in the way $\overline{C}$ was used in Phase 2. To determine the target value, observe that after

interchange, the value of C_1 will decrease to $C_1' = C_1 - \theta$, and that of C_h will increase to $C_h' = C_h + \theta$, so that $\mid C_1' - C_h' \mid = \mid C_1 - C_h - 2\theta \mid = \mid d - 2\theta \mid$, and this value is minimized for $\theta = d/2$. It is not necessary to enumerate all possible interchange pairs. Instead, we take advantage of the interval structure induced by the job processing times, as the search is restricted to job pairs belonging to two intervals located approximately $d/2$ units apart. Formally, the procedure can be described as follows.

Step 1. Set $h := m$.

Step 2. Set $t := r$, $f_t := C_1$ and $d := C_1 - C_h$. If $\overline{p} - d/2 \leq \underline{p}$, set $s := r$. Otherwise, let s be the index of the interval to which $\overline{p} - d/2$ belongs.

Step 3. If there exist two jobs J_j and $J_{j'}$ with $p_j \in I_t, p_{j'} \in I_s$, such that J_j is assigned to $M_1, J_{j'}$ is assigned to M_h, and $0 < p_j - p_{j'} < d$, compute $f_t = \min_{p_j \in I_t, p_{j'} \in I_s} \left\{ \max \left[C_1 - p_j + p_{j'}, C_h + p_j - p_{j'} \right] \right\}$. If $s = 1$, set $\underline{t} := t$ and go to Step 5.

Step 4. Set $s := s - 1$, $t := t - 1$ and go to Step 3.

Step 5. Set $f^* := \min_{\underline{t} \leq t \leq r} \{f_t\}$. If $f^* < C_1$, interchange the two jobs J_j and $J_{j'}$ yielding f^* and go to Phase 2. If $f^* = C_1$, set $h := h - 1$. If $h > 1$, go to Step 2. If $h = 1$, stop. ■

It is worth observing that the partitioning of jobs into r intervals has a significant impact on the computation of f_t in Step 3, and hence on the overall performance of Phase 3. Assuming uniformly distributed processing times, the average number of jobs per machine with processing time in a given interval is approximately equal to n/rm, which implies that around $(n/rm)^2$ pairs of jobs

are considered in the determination of f_t for each value of t. Therefore, the computation of f^* requires approximately $2r(n/rm)^2$ comparisons, as opposed to $2(n/m)^2$ when no partitioning is used.

Considering the fact that the heuristic may iterate an unspecified number of times between Phases 2 and 3, it is not possible to derive a simple worst-case complexity bound for it. This is often the case of algorithms designed with the aim of achieving good empirical performance rather than theoretical efficiency. We may also show that the worst-case performance bound for the heuristic can be quite large by considering a family of pathological examples. For any given number of machines $m \geq 3$, consider a set of $(m-1)$ jobs with processing times $p_j = k + j/m$ $(j = 1, ..., m-1)$, and set $r := m-1$ so that there is a single job in each interval I_t. For k large enough $(k \geq m - 1/2)$, one will have $\overline{C} \leq k$. When the heuristic is applied, Phase 1 will assign all jobs to M_1 and no job movement will be performed either by Phase 2 (because all p_j's are greater than $\overline{C}$) or Phase 3 (because there are no jobs on M_2 to M_m), and the heuristic will terminate with all jobs on M_1. Obviously, an optimal solution is to assign a single job to each of machines M_1 to M_{m-1}. The performance ratio is thus equal to $\sum_{j=1}^{m-1} p_j / p_{m-1}$ $= [(m-1)k + (m-1)/2]/[k + (m-1)/m]$, which tends to $(m-1)$ as k tends to infinity. The empirical performance of the heuristic on random problems is, however, very good. This will be illustrated in Section 3.

3. Computational Results

The 3–PHASE heuristic was compared with three other well known algorithms:

1) the classical $0/1$ interchange heuristic of Finn and Horowitz [15];
2) the improved $0/1$ interchange heuristic of Langston [17];
3) the LPT heuristic of Graham [8].

The $0/1$ interchange heuristic and its improved version are known to have very good computational performances. The LPT heuristic was selected as it is often used as a benchmark for test problems. We did not compare 3–PHASE with the Multifit algorithm of Coffman, Garey and Johnson [9] as this algorithm does not seem to perform as well as LPT [8].

The four heuristics were compared between themselves, and the solution values produced by these algorithms were also compared with the optimal values obtained by solving a sequence of bin packing problems within a bisection search scheme (see, e.g., Dell'Amico and Martello [6]). For the solution of the bin packing problems, we used the MTP code of Martello and Toth [18].

Tests were made on problems with m = 5,10,25 and n = 10,50,100,500,1000, and processing times in the intervals [1, 100], [1, 1000] and [1, 10000]. Uniform, as well as truncated exponential and normal distributions were tested, but they exhibit basically the same behavior. All reported results will be for uniformly distributed processing times. We first analyzed the effect of r, the number of intervals for the processing times, on the performance of the algorithm by running several test problems for $r = 1, ..., 20$. Basically, a low value of r will produce a

good Phase 1 solution, and few passes of Phases 2 and 3; Phase 3 will, however, be computationally expensive. Larger values of r yield worse initial solutions, but will speed up the improvement phases. Overall, we found empirically that the "best" value of r is related to the ratio (n/m). The "recommended values" r^* given in Table 1 usually yield a solution value and a computing time very close to the best among all values of r for a given instance.

Table 1. Recommended values r^*

n/m	r^*
(1, 10)	2
[10, 50)	5
[50, 100)	10
[100, 200)	15
[200, 400)	20

All problems were then run with $r := r^*$. We report in Tables 2 and 3 results for $p_j \in [1, 100]$ and $p_j \in [1, 10000]$. (Results for $p_j \in [1, 1000]$ fall somewhere in between and are not reported here.) Unless otherwise indicated, all entries are average values over 10 problems. The only exceptions are when averages are taken over the number of problems solved to optimality and this number is less than 10. The number of successful problems out of 10 is then shown in brackets in columns 3 and 7 of Table 3. The variables used in Tables 2 and 3 are defined as follows:

z^* : global optimum;

z : best solution value found by the 3-PHASE algorithm;

z_1 : best solution value found by the 0-1 interchange algorithm [15];

z_2 : best solution value found by the improved 0-1 interchange algorithm [17];

z_3 : best solution value found by the LPT algorithm [8].

The four heuristics were coded in C and all tests were executed on a Sun Sparc 2 station. The MTP bin packing program used as a base for the exact algorithm is written according to PFORT, a portable subset of 1966 ANSI Fortran [19]. A time limit of 1000 seconds was imposed for the solution of any instance.

Results reported in Tables 2 and 3 indicate that the 3–PHASE algorithm produces solution values that are almost always well within 1% of the optimum, when not optimal (see column 3). Moreover, our algorithm always yields average solution values at least as good as those obtained with any of the three alternative heuristics used for the comparison: this is indicated by the non-negative ratios in columns 4, 5 and 6. The 3–PHASE algorithm is also always quicker than these heuristics, sometimes by one order of magnitude. In short, the proposed algorithm produces optimal or quasi-optimal solutions in relatively short times and its empirical performance dominates that of the three benchmark heuristics.

Table 2. Computational results for $p_j \in [1, 100]$

m	n	$\left(\frac{z-z^*}{z^*}\right)$	$\left(\frac{z_1-z}{z}\right)$	$\left(\frac{z_2-z}{z}\right)$	$\left(\frac{z_3-z}{z}\right)$	CPU time (seconds)				
						z^*	z	z_1	z_2	z_3
5	10	0.018	0.027	0.027	0.000	0.003	0.002	0.030	0.038	0.033
	50	0.000	0.008	0.008	0.005	0.003	0.002	0.032	0.037	0.037
	100	0.000	0.001	0.001	0.001	0.008	0.001	0.048	0.037	0.035
	500	0.000	0.000	0.000	0.000	0.024	0.022	0.108	0.110	0.063
	1000	0.000	0.000	0.000	0.000	0.063	0.063	0.215	0.210	0.153
10	50	0.002	0.025	0.025	0.013	0.013	0.003	0.033	0.035	0.035
	100	0.002	0.007	0.007	0.003	0.008	0.005	0.043	0.037	0.033
	500	0.000	0.001	0.001	0.000	0.014	0.015	0.128	0.140	0.053
	1000	0.000	0.000	0.000	0.000	0.034	0.040	0.293	0.308	0.102
25	50	0.011	0.008	0.008	0.000	0.017	0.008	0.035	0.040	0.030
	100	0.003	0.004	0.004	0.021	0.023	0.013	0.040	0.040	0.042
	500	0.000	0.002	0.002	0.001	0.026	0.018	0.160	0.158	0.057
	1000	0.000	0.000	0.000	0.000	0.041	0.045	0.445	0.455	0.093

Table 3. Computational results for $p_j \in [1, 10000]$

m	n	$\left(\frac{z-z^*}{z^*}\right)$	$\left(\frac{z_1-z}{z}\right)$	$\left(\frac{z_2-z}{z}\right)$	$\left(\frac{z_3-z}{z}\right)$	CPU time (seconds)				
						z^*	z	z_1	z_2	z_3
5	10	0.013	0.036	0.036	0.003	0.005	0.001	0.033	0.032	0.032
	50	0.000 (7)	0.006	0.006	0.005	111.402 (7)	0.005	0.036	0.033	0.029
	100	0.000 (8)	0.002	0.002	0.001	22.331 (8)	0.002	0.044	0.046	0.031
	500	0.000	0.000	0.000	0.000	0.912	0.035	0.183	0.187	0.067
	1000	0.000	0.000	0.000	0.000	0.053	0.054	0.632	0.628	0.148
10	50	0.004 (5)	0.026	0.026	0.021	239.700 (5)	0.001	0.042	0.025	0.038
	100	0.000 (1)	0.007	0.007	0.004	2.333 (1)	0.001	0.050	0.050	0.017
	500	0.000 (7)	0.000	0.000	0.000	1.238 (7)	0.042	0.183	0.175	0.050
	1000	0.000 (9)	0.000	0.000	0.000	0.066 (9)	0.067	0.604	0.608	0.104
25	50	0.003	0.048	0.048	0.004	0.010	0.013	0.032	0.035	0.033
	100	0.001 (1)	0.047	0.047	0.023	0.100 (1)	0.017	0.033	0.044	0.050
	500	– (0)	0.002	0.002	0.001	— (0)	0.045	0.175	0.160	0.087
	1000	0.000 (4)	0.000	0.000	0.000	0.142 (4)	0.072	0.452	0.447	0.134

Acknowledgements

The work of Paulo M. França was supported in part by Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), and by Conselho Nacional de Desenvolvimento Cientifíco e Tecnológico (CNPq), Brazil. Michel Gendreau and Gilbert Laporte were funded by the Canadian Natural Sciences and Engineering Research Council under grants OGP0038816 and OGP0039682. Felipe M. Müller was partially supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Brazil.

References

1. R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* 5, 287–326 (1979).
2. E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, Sequencing and scheduling: Algorithms and complexity. Report BS-R8909, Center for Mathematics and Computer Science, Amsterdam (1989).
3. T.C.E. Cheng and C.C.S. Sin, A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research* 47, 271–292 (1990).
4. M.R. Garey and D.S. Johnson, Strong NP-completeness results: motivation, examples and implications. *Journal of the Association for Computing Machinery* 25, 499–508 (1978).
5. J. Blazewicz, Selected topics in scheduling theory. *Annals of Discrete Mathematics* 31, 1–60 (1987).

6. M. Dell'Amico and S. Martello, Optimal scheduling of tasks on identical parallel processors. Research report OR/90/7, DEIS, University of Bologna (1990).

7. P. Bratley, M. Florian and P. Robillard, Scheduling with earliest start and due date constraints on multiple machines. *Naval Research Logistics Quarterly* 22, 165–173 (1975).

8. R.L. Graham, Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics* 17, 416–429 (1969).

9. E.G. Coffman Jr., M.R. Garey and D.S. Johnson, An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing* 7, 1–17 (1978).

10. D.K. Friesen, Tighter bounds for LPT scheduling on uniform processors. *SIAM Journal on Computing* 16, 554–560 (1987).

11. D.K. Friesen and M.A. Langston, Evaluation of a MULTIFIT-based scheduling algorithm. *Journal of Algorithms* 7, 35–59 (1986).

12. S.K. Sanhi, Algorithms for scheduling independent tasks. *Journal of the Association for Computing Machinery* 23, 116–127 (1976).

13. D.S. Hochbaum and D.B. Shmoys, Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the Association for Computing Machinery* 34, 144–162 (1987).

14. J.Y.-T. Leung, Bin packing with restricted piece sizes. *Information Processing Letters* 31, 145–149 (1989).

15. G. Finn and E. Horowitz, A linear time approximation algorithm for multiprocessor scheduling. *BIT* 19, 312–320 (1979).

16. E.G. Coffman Jr., M.R. Garey and D.S. Johnson, Approximation algorithms for bin-packing — An updated survey. In *Algorithm Design for Computer System Design*, 49–106, G. Ausiello, M. Lucertini and P. Serafini (eds.). Springer, Vienna (1984).

17. M.A. Langston, Improved 0/1 interchange scheduling. *BIT* 22, 282–290 (1982).

18. S. Martello and P. Toth, *Knapsack Problems*. Wiley, Chichester (1990).

19. B.F. Ryder and A.D. Hall, The PFORT verifier. Computer Science Report 2, Bell Laboratories (1981).